> An **MCP Server** is a **specialized server** that provides **tools, resources, and context** to help an LLM perform actions—like fetching database records, editing files, or interacting with APIs.

Without MCP, LLMs are like brilliant minds trapped in a box. They can explain how to cook lasagna but can't turn on the stove. With MCP, they finally get access to the kitchen.

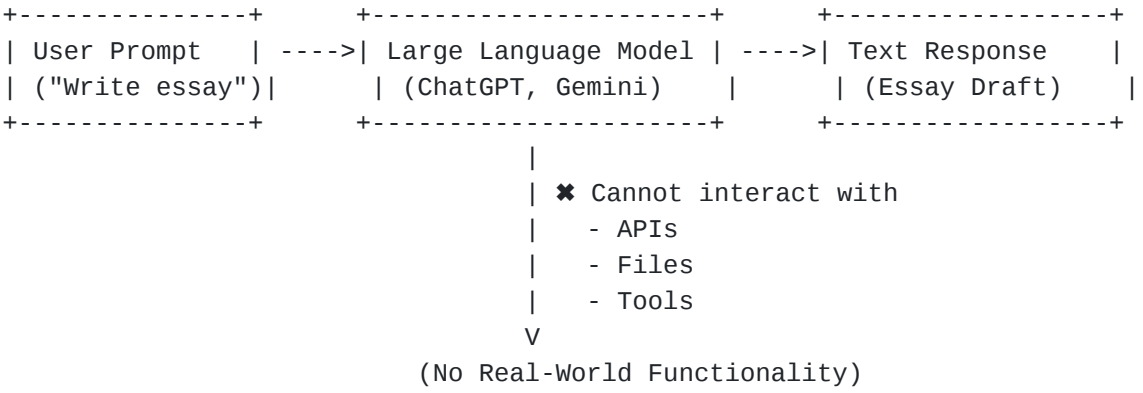
⚠️ The LLM Limitation

LLMs are phenomenal at generating text, but here's what they **can't do out-of-the-box**:

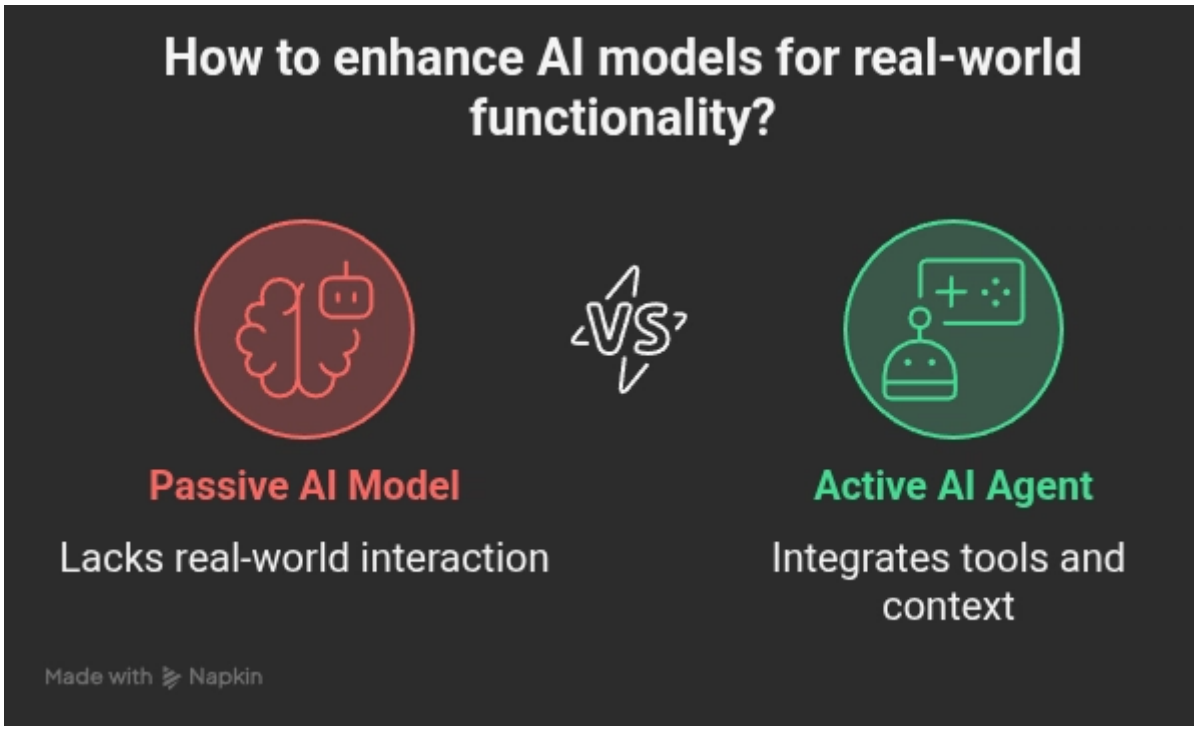
- Send an email
- Execute a Python script
- Retrieve files from a local system
- Run database queries
- Edit a Figma design
- Trigger a Stripe payment

Why? Because they don't inherently have access to **context** or **actionable interfaces**.

🧩 The Missing Link (Visualized)



This is where **MCP Servers step in**—they add **tools, context, and protocols** that transform a passive AI model into an active digital agent.



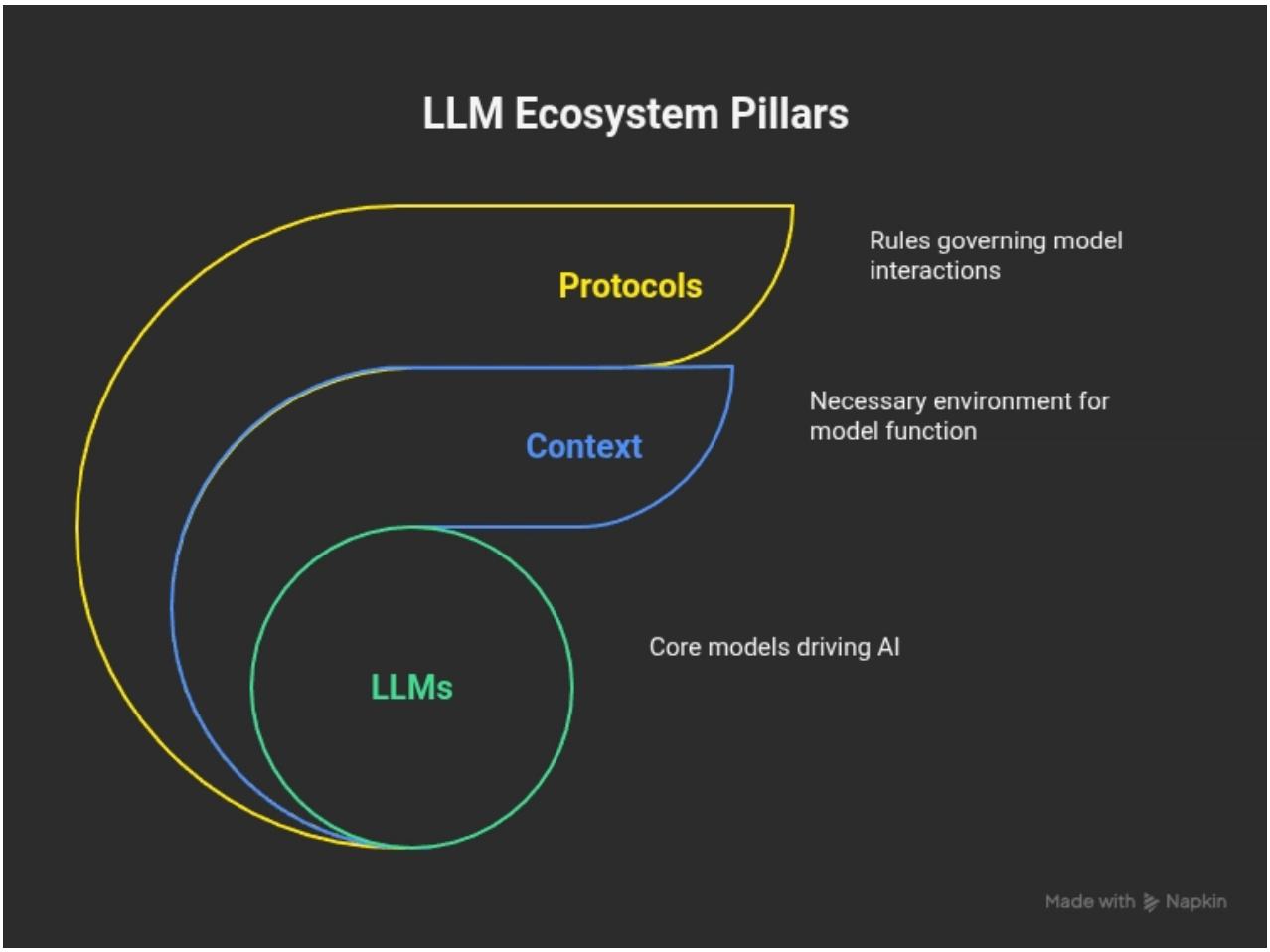
🔗 MCP: Model Context Protocol Demystified

MCP isn't software—it's a protocol. Just like HTTP governs how browsers talk to websites, MCP governs **how LLMs talk to tools**.

It's an attempt to standardize AI integration, providing a shared language between LLMs and external systems.

MCP Has Three Core Pillars:

1. **Models** – The LLMs themselves
2. **Context** – The information or environment the model needs
3. **Protocols** – The standardized rules of interaction



Let’s break these down.

🧠 1. Models: More Than Just Chatbots

MCP is built for all kinds of models:

- **Text Models** (e.g., ChatGPT, Claude)
- **Image Models** (e.g., for editing or generating images)
- **Video Models** (e.g., for rendering or scene understanding)

It provides a **universal interface** for any model to interact with external systems—regardless of the media type.

🧩 2. Context: The Secret Ingredient

LLMs are context-hungry.

For example, if you ask an LLM:

> “Fix the bug in my repo”
> It needs to know:

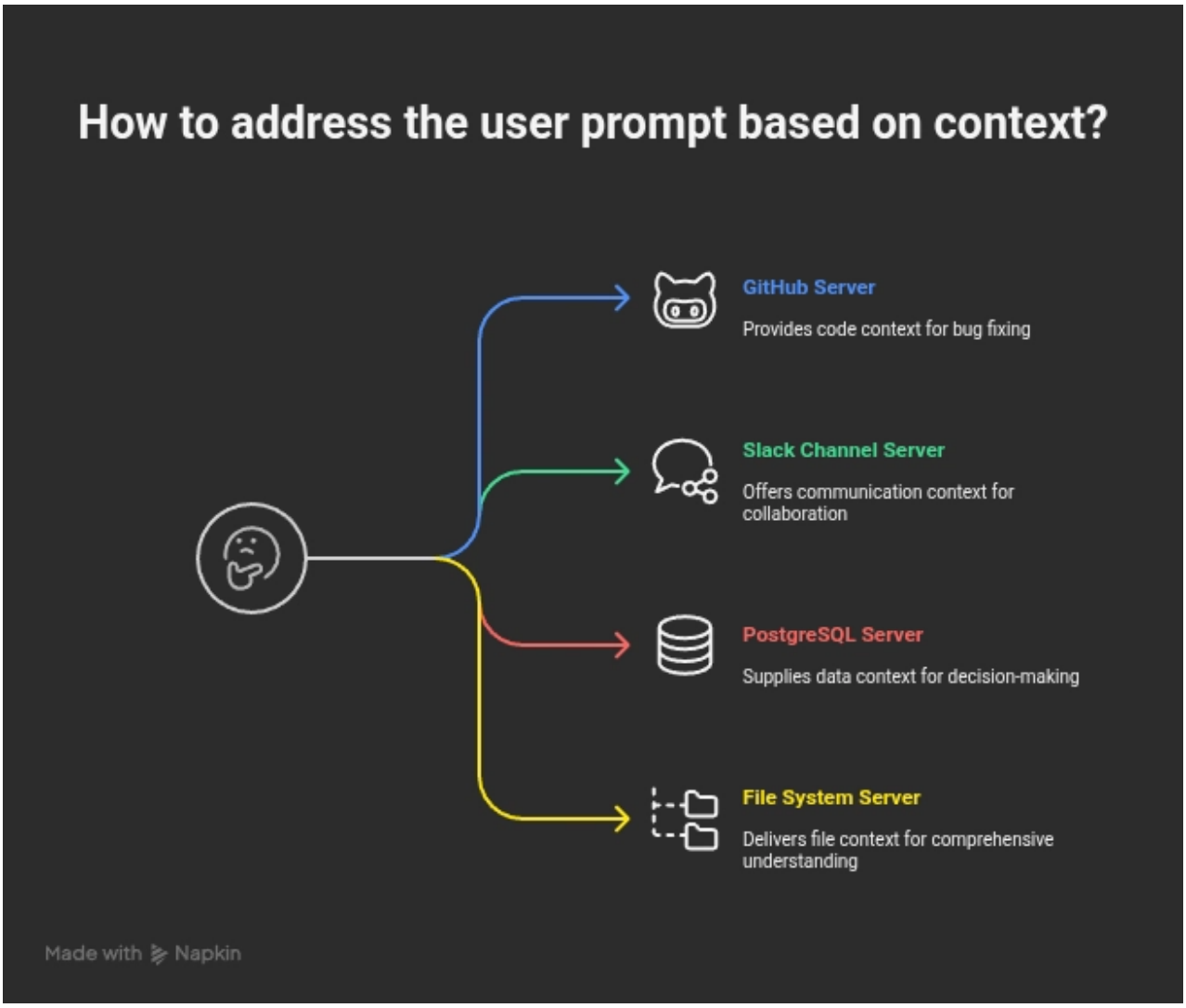
- Which repo?

- What files?
- What error logs?

MCP Servers provide that missing context—**from GitHub issues, Slack channels, code files, or databases**.

🔍 Visual: LLM + MCP Context

```
+--------------+      +------------------+      +----------------------+
| User Prompt  | ---->| Large Language   | ---->| Action/Decision      |
| ("Fix bug in |      | Model (LLM)      |      | Based on Full Context|
| repo/slack") |      +------------------+      +----------------------+
                ^              |
                |              V
+---------------+--------------------------+
|      MCP Servers (Provide Context)       |
|   - GitHub Server                        |
|   - Slack Channel Server                 |
|   - PostgreSQL Server                    |
|   - File System Server                   |
+------------------------------------------+
```

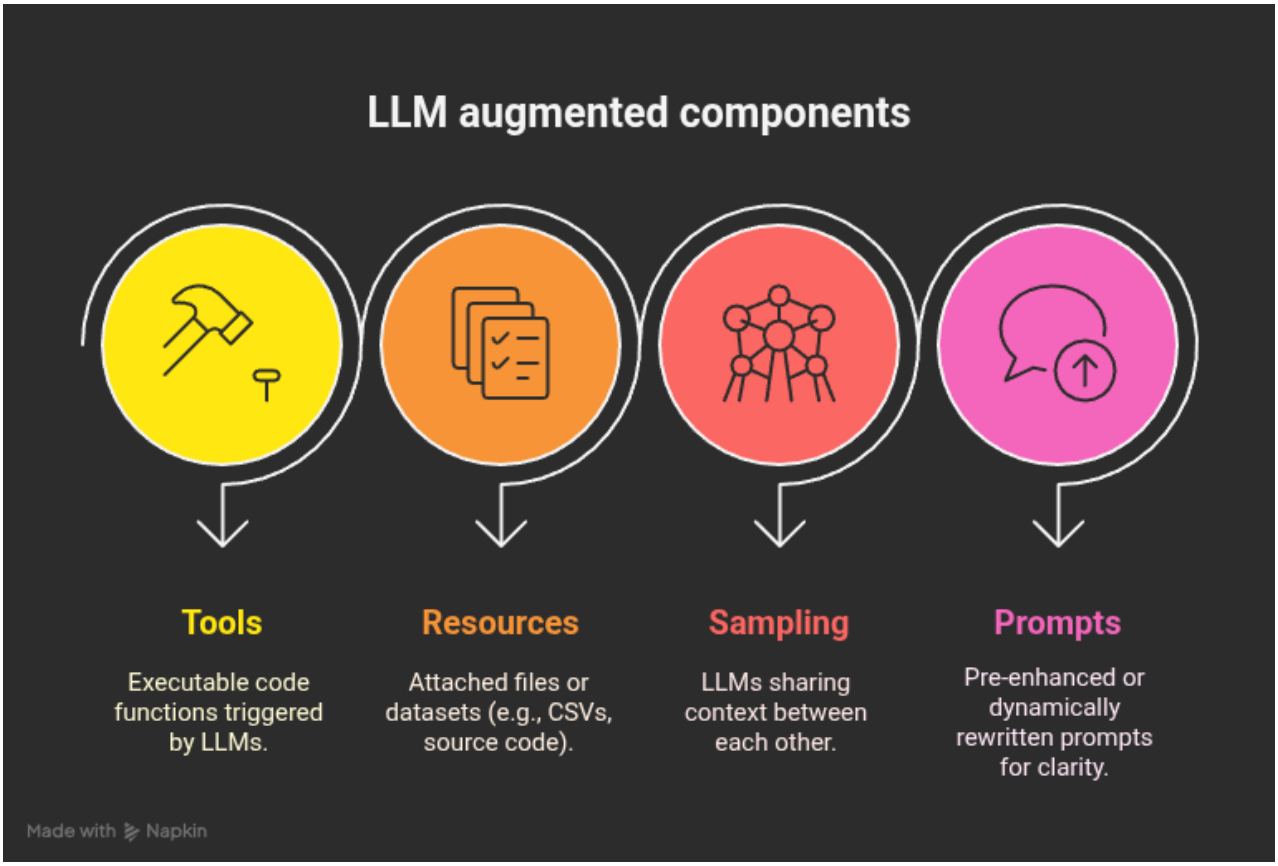

How to address the user prompt based on context?
GitHub Server
Provides code context for bug fixing
Slack Channel Server
Offers communication context for collaboration
PostgreSQL Server
Supplies data context for decision-making
File System Server
Delivers file context for comprehensive understanding
Made with Napkin

🛠️ How Context Is Delivered

MCP defines four mechanisms to deliver context and capability:

1. **Tools (Functions)** – Executable code functions triggered by LLMs
2. **Resources** – Attached files or datasets (e.g., CSVs, source code)
3. **Sampling** – LLMs sharing context between each other
4. **Prompts** – Pre-enhanced or dynamically rewritten prompts for clarity

> These aren't magic—they're *real functions*, APIs, and files running on a real server.



🔌 3. Protocols: Making It All Work

The "Protocol" in MCP defines **how requests, capabilities, and tools are exposed to the model**.

Think of it like API documentation—but for LLMs. It includes:

- How tools are defined
- How models can "reflect" on what a server offers
- The format for making requests or returning results

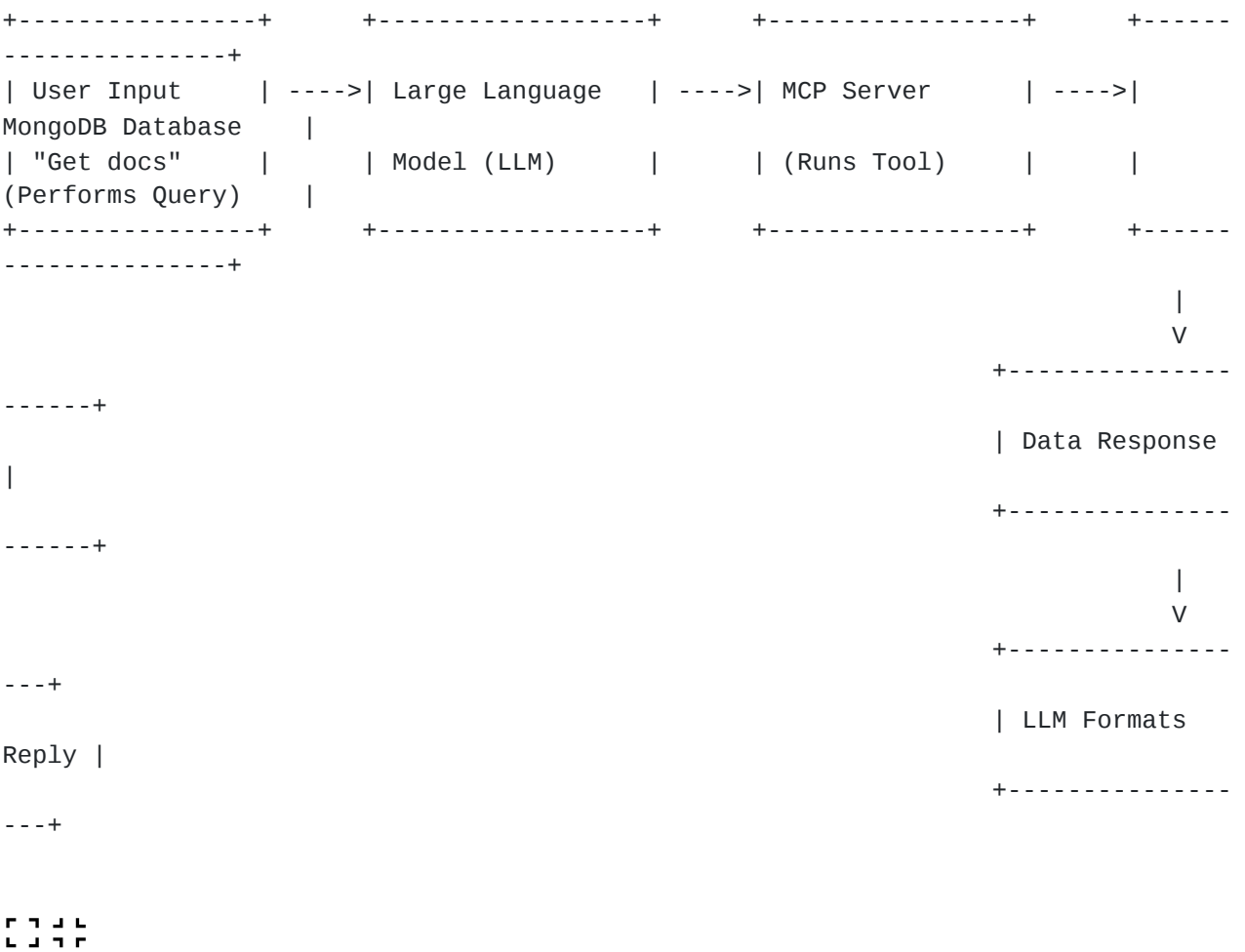
This is what allows the LLM to intelligently ask:

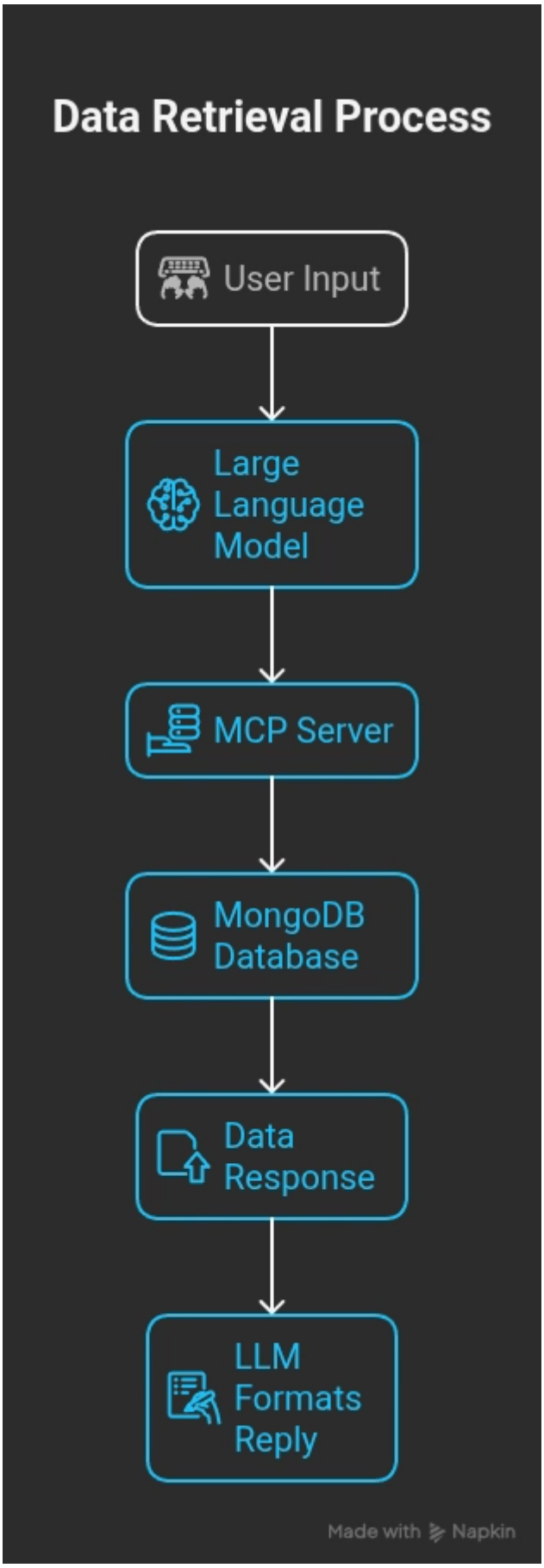
> “What tools are available?”
> And then invoke them—without hardcoding every step.

🛠️ How an MCP Server Works (Step-by-Step)

Let’s take a MongoDB example:

🔁 Flow Overview





✔️ Example Functions on an MCP Mongo Server:

```
// Node.js example
function connectToMongo() { ... }
function getDocuments() { ... }
function insertDocument() { ... }
```

⛶

LLMs don't write these—they just **call them** when needed, thanks to MCP protocols.

🛍️ The Emerging MCP Ecosystem

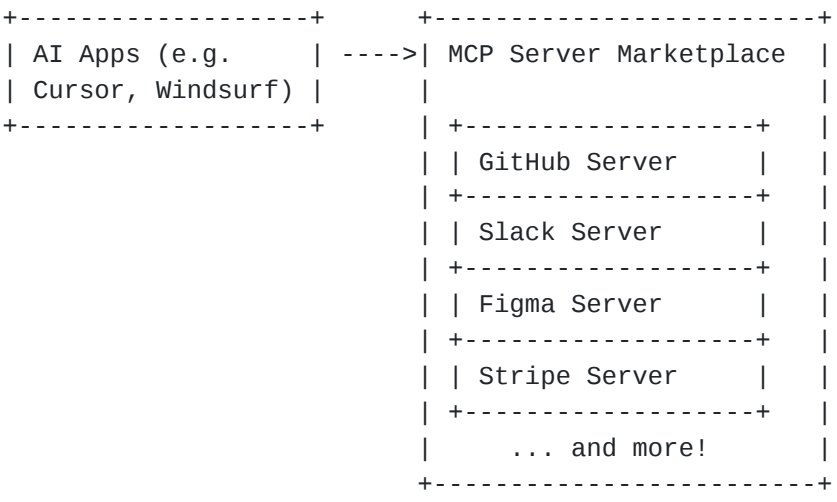
As MCP adoption grows, a **marketplace of plug-and-play MCP servers** is emerging.

🔌 Current Platforms:

- **Windsurf**: Add GitHub, Slack, PostgreSQL, Figma, Stripe, etc. via MCP servers.
- **Puls.com**: Offers MCP servers for automation, file systems, browsers, and more.

You simply provide access tokens or credentials, and you're good to go.

🔄 Diagram: MCP Server Marketplace



⛶

🔮 Why MCP Servers Matter (And Why You Should Care)

MCP Servers aren't just a backend convenience—they're **a paradigm shift**:

✔️ For Developers:

- Build MCP servers in **Python, Node.js, or Java**
- Define tools and operations as normal functions
- Integrate LLMs into any environment—web, design, databases, even games

✔️ For the AI Ecosystem: